Math. Struct. in Comp. Science (2005), *vol.* 15, *pp.* 959–968.
doi:10.1017/S0960129505004883 Printed in the United Kingdom

Matching with multiplication and exponentiation (extended abstract)†

BERNARD LANG

IRIA – Laboria, Domaine de Voluceau, Rocquencourt, 78150 LE CHESNAY, France‡

Received 4 April 2003; revised 3 July 2004

We develop a matching algorithm for an equational theory with multiplication, exponentiation and a unit element. The algorithm is proved consistent, complete and minimal using techniques based on initial algebras.

1. Introduction

The interest in unification, that is, the resolution of equations in a term language, has been strongly motivated by automated theorem provers (Robinson 1965). Efficiency considerations have then led to the incorporation of equational axioms into the unification algorithms, that is, to unification in equational theories (Plotkin 1972; Slagle 1974).

Given a system $\{t_i = t'_i \mid 1 \leqslant i \leqslant n\}$ of n equations, the problem is to find some substitution σ on the variables occuring in the terms t_i and t'_i such that for all indices i we have the equality $\sigma t_i = \sigma t'_i$, or possibly just an equivalence modulo some equational axioms.

In this paper we will consider more particularly the problem of matching, that is, the resolution of equations in which the right-hand side contains no variables. This simpler problem has numerous applications in symbolic computation and, in particular, in program manipulation and transformation (Darlington and Burstall 1973).

So far unification or matching algorithms have been developed only for a small number of equational theories whose axioms are some combination of associativity, commutativity, 'idempotence' and existence of a unit element (Kühner *et al.* 1977; Plotkin 1972; Siekmann 1975; Stickel 1975). We must also mention a small 'arithmetic evaluation' theory by Plotkin (Plotkin 1972), the work of Huet on unification in typed lambda-calculus (Huet 1975), and a fast algorithm for unification in a free theory (Paterson and Wegman 1976).

The only general framework for the study of unification has been given by Plotkin in Plotkin (1972), and has also been used by others (Huet 1975; Kühner *et al.* 1977). Given an equational theory, one has to define for every term t an effectively computable equivalent normal form $N(t)$, which must be unique in every class of terms equivalent modulo the axioms of the theory. Unification or matching algorithms can then be defined for normal forms only.

† Written May 1978 – see the historical note at the end of the paper.

‡ The current coordinates of the author are:
INRIA, B.P. 105, 78153 Le Chesnay CEDEX, France – Bernard.Lang@inria.fr

We believe that this essentially syntactic approach presents some drawbacks: there may be no natural normal form in the theory considered (for example, commutative functions) or, when there is one, it is not usually preserved by substitution, and much overhead computation is caused by renormalisation during unification. Finally, the definition of a normal form and of the corresponding normalisation preocedure may be a tedious task with little relevance to the problem at hand (this is the case for the theory studied in this paper).

The alternative we use is more semantic in nature. We study the matching (or unification) problem directly in an algebra that is initial in the category of all algebras satisfying the axioms. This initial algebra is defined as the quotient of the free algebra of terms by a congruence induced by the axioms.

One advantage of this approach is that many definitions and results can be stated and/or proved once and for all, independently of the axioms used. A second advantage is that all reasonning is simpler when conducted directly in the theory concerned, rather than in some syntactic representation domain. Third, this approach is always possible since the initial algebra always exists with equational axioms.

However, one must be careful that an algorithm defined in a semantic domain must actually be carried out on syntactic representations of the domain elements. This is why all functions on the quotient algebra $\mathscr{E}$ that we use are defined by means of morphisms, thus automatically yielding an identical definition on the free algebra $\mathscr{T}$ of terms representing syntactically the elements of $\mathscr{E}$. For example, the variable set $\mathscr{V}e$ of an element e of $\mathscr{E}$ has a general definition as a possibly infinite intersection, and thus is not computationally usable. For each specific theory we give an equivalent definition in terms of morphisms.

The bulk of the paper is the application of the above approach to the development of a matching algorithm in a theory with multiplication, exponentiation and a unit element, denoted, respectively, by '$\times$', '$\uparrow$' and '1', and satisfying the following system $\mathscr{A}$ of axioms:

$$\mathscr{A} : \begin{cases} \mathscr{A}_1 : (x \times y) \times z = x \times (y \times z) & \text{associativity} \\ \mathscr{A}_2 : (x \times y) \uparrow z = (x \uparrow z) \times (y \uparrow z) & \text{left distributivity} \\ \mathscr{A}_3 : x \uparrow (y \times z) = (x \uparrow y) \uparrow z & \text{right currying} \\ \mathscr{A}_4 : x \times 1 = x & \\ \mathscr{A}_5 : 1 \times x = x & \\ \mathscr{A}_6 : x \uparrow 1 = x & \\ \mathscr{A}_7 : 1 \uparrow x = 1. & \end{cases}$$

The matching algorithm is based on the decomposition of expressions into base and exponent (in the usual sense, that is, a and b, are, respectively, the base and exponent of $a \uparrow b$). This decomposition is used to reduce our matching problem to that of matching sequences in a free monoid, which we know how to solve (Plotkin 1972; Siekmann 1975).

We prove this matching algorithm to be consistent, complete and minimal.

A final short section of the paper is devoted to efficiency considerations and suggestions for the implementation of the algorithm.

There are at least three sorts of interpretations for the above equational theory. First, there are the usual arithmetic interpretations (with integer, reals,...). Next there are boolean interpretations such as, for example, the interpretation of '$\times$', '$\uparrow$' and

https://doi.org/10.1017/S0960129505004883 Published online by Cambridge University Press

'1', respectively, by conjunction, (reverse) implication and true. Finally, we have 'type' interpretations on sets or domains (in the sense of Scott (Scott 1977)) when arbitrary currying of functions is allowed: then '$\times$', '$\uparrow$' and '1' denote, respectively, the cartesian product, the function space operator and the singleton set or domain (considered unique).

2. Algebraic framework

Given a denumerable set $\mathscr{F}$ of function symbols graduated by an arity and an infinitely denumerable set $\mathscr{V}$ of variables, we define $\mathscr{F}$-algebras, the free $\mathscr{F}$-algebra $\mathrm{M}(\mathscr{F},\mathscr{V})$ on $\mathscr{V}$, interpretations of $\mathscr{F}$ and valuations of $\mathscr{V}$, morphisms of $\mathscr{F}$-algebras, the variable set $\mathscr{V}t$ of a term t of $\mathrm{M}(\mathscr{F},\mathscr{V})$, the size $|t|$ of the term t, and the quotient of an $\mathscr{F}$-algebra by a congruence.

The domain of an endomorphism μ on $\mathrm{M}(\mathscr{F},\mathscr{V})$ is denoted $\mathscr{D}_\mu$ and is defined by $\mathscr{D}_\mu = \{x \in \mathscr{V} \mid \mu x \neq x\}$. A substitution on $\mathrm{M}(\mathscr{F},\mathscr{V})$ is an endomorphism with finite domain. We use $\Sigma(\mathscr{F},\mathscr{V})$ to denote the set of substitutions on $\mathrm{M}(\mathscr{F},\mathscr{V})$.

Given a system $\mathscr{A}$ of equational axioms on $\mathrm{M}(\mathscr{F},\mathscr{V})$, we associate to it a congruence $\sim_{\mathscr{A}}$ in any $\mathscr{F}$-algebra. We define $\mathscr{F}/\mathscr{A}$-algebras (that is, $\mathscr{F}$-algebras satisfying $\mathscr{A}$) and the initial $\mathscr{F}/\mathscr{A}$-algebra, denoted $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$. The elements of $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$ are called *expressions* and are the congruence classes modulo $\mathscr{A}$ of the terms in $\mathrm{M}(\mathscr{F},\mathscr{V})$. An element t of an expression e is called a (syntactic) *representation* of e; we write $t \in e$ or $e = [t]_{\mathscr{A}}$. The variable set of an expression is defined as $\mathscr{V}e = \bigcap_{t\in e} \mathscr{V}t$. We note that structural induction is valid in $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$.

The domain of an endomorphism μ on $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$ is denoted $\mathscr{D}_\mu$ and is defined by $\mathscr{D}_\mu = \{x \in \mathscr{V} \mid x \notin \mu[x]_{\mathscr{A}}\}$. A substitution on $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$ is an endomorphism with finite domain. The set of substitutions on $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$ can be identified with the quotient $\Sigma(\mathscr{F},\mathscr{V})/\mathscr{A}$ of $\Sigma(\mathscr{F},\mathscr{V})$ by an equivalence suitably defined from $\mathscr{A}$. Thus, every substitution σ in $\Sigma(\mathscr{F},\mathscr{V})/\mathscr{A}$ has at least one syntactic representation κ in $\Sigma(\mathscr{F},\mathscr{V})$; we also write $\kappa \in \sigma$ or $\sigma = [\kappa]_{\mathscr{A}}$. The variable set of a substitution σ is $\mathscr{V}\sigma = \bigcup_{x\in\mathscr{D}_\sigma} \mathscr{V}(\sigma[x]_{\mathscr{A}})$. The restriction of a substitution σ to the subset $\mathscr{W}$ of $\mathscr{V}$ is denoted $\sigma \restriction \mathscr{W}$. The *identity substitution* has an empty domain and is conventionally denoted by $\varnothing$.

Several lemmas relate expressions and substitutions to their representations. If σ and σ' are in $\Sigma(\mathscr{F},\mathscr{V})/\mathscr{A}$ and e is an expression in $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$, we have, for example,

$$t \in e \text{ and } \kappa \in \sigma \implies \kappa t \in \sigma e$$

$$\kappa \in \sigma \text{ and } \kappa' \in \sigma' \implies \kappa \circ \kappa' \in \sigma \circ \sigma'$$

$$\mathscr{D}\sigma = \bigcap_{\kappa\in\sigma} \mathscr{D}\kappa \quad \text{and} \quad \mathscr{V}\sigma = \bigcap_{\kappa\in\sigma} \mathscr{V}\kappa.$$

Thus it is often convenient, and harmless, to denote an expression or a substitution by one of its representations.

We define on $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$ the *approximation* preorder, denoted $\leqslant$, by

$$e \leqslant e' \quad \text{iff} \quad \exists \sigma \in \Sigma(\mathscr{F},\mathscr{V})/\mathscr{A} \;.\; e' = \sigma e.$$

https://doi.org/10.1017/S0960129505004883 Published online by Cambridge University Press

This preorder is extended to $\Sigma(\mathscr{F},\mathscr{V})/\mathscr{A}$ by defining

$$\sigma \leqslant \sigma' \quad \text{iff} \quad \exists \rho \in \Sigma(\mathscr{F},\mathscr{V})/\mathscr{A} \,.\; \sigma' = \rho \circ \sigma.$$

Of course, for any expression e we have $\sigma \leqslant \sigma' \Longrightarrow \sigma e \leqslant \sigma' e$.

Several useful results can be established independently of the system of axioms considered. Let e be an expression in $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$, let σ, σ' and ρ be substitutions in $\Sigma(\mathscr{F},\mathscr{V})/\mathscr{A}$ and let $\mathscr{W}$ be a subset of $\mathscr{V}$. Then we have:

(a) $\mathscr{D}(\sigma \circ \sigma') \subseteq \mathscr{D}\sigma \cup \mathscr{D}\sigma'$.
(b) $\mathscr{V}e \subseteq \mathscr{W} \implies (\sigma \restriction \mathscr{W})e = \sigma e$.
(c) $(\mathscr{V}(\sigma \restriction \mathscr{W}) \cap \mathscr{D}\rho) \subseteq \mathscr{W} \implies (\rho \circ \sigma) \restriction \mathscr{W} = (\rho \restriction \mathscr{W}) \circ (\sigma \restriction \mathscr{W})$.
(d) $\sigma \leqslant \sigma' \implies \sigma \circ \rho \leqslant \sigma' \circ \rho$.
(e) $\mathscr{V}(\sigma \restriction \mathscr{W}) = \varnothing$ and $\sigma \leqslant \sigma' \implies \sigma \restriction \mathscr{W} \leqslant \sigma' \restriction \mathscr{W}$.
(f) $\sigma < \sigma'$ and $\mathscr{V}\sigma = \varnothing \implies \sigma = \sigma' \restriction \mathscr{D}\sigma$ and $\mathscr{D}\sigma \subset \mathscr{D}\sigma'$.
(g) $\sigma \leqslant \sigma' \restriction \mathscr{W}$ and $\mathscr{V}\sigma = \varnothing \implies \sigma \leqslant \sigma'$.

A matching pair on $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$ is a pair $\langle e, e'\rangle$ of expressions in $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$ such that $\mathscr{V}e' = \varnothing$.

If $\mathscr{P}$ is a set of matching pairs on $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$, a substitution σ is a *match* of $\mathscr{P}$, or is *consistent* with $\mathscr{P}$, iff we have $e' = \sigma e$ for every pair $\langle e, e'\rangle$ in $\mathscr{P}$. We use $\mathscr{M}\mathscr{P}$ to denote the set of all matches of $\mathscr{P}$.

A set S of substitutions in $\Sigma(\mathscr{F},\mathscr{V})/\mathscr{A}$ is said to be:

— *consistent* with $\mathscr{P}$ iff every substitution in σ in S is consistent with $\mathscr{P}$;
— *complete* with respect to $\mathscr{P}$ iff every match σ' of $\mathscr{P}$ is approximated by a substitution in S;
— *minimal* iff no substitution in S is approximated by a different one.

A *matching algorithm* for $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$ is an algorithm that, given a set $\mathscr{P}$ of matching pairs on $\mathrm{M}(\mathscr{F},\mathscr{V})/\mathscr{A}$, computes from $\mathscr{P}$ a set S of matches of $\mathscr{P}$.

The algorithm is said to be *complete* (respectively, *minimal*) iff the computed set S is always complete with respect to $\mathscr{P}$(respectively, minimal).

For convenience, we add to the set of sets of matching pairs the special symbol Ⓕ, which we consider as a set of matching pairs without matches, that is, $\mathscr{M}Ⓕ = \varnothing$.

If $\mathscr{P}$ is a set of matching pairs, we define the variable set of $\mathscr{P}$ to be

$$\mathscr{V}\mathscr{P} = \begin{cases} \bigcup_{\langle e,e'\rangle \in \mathscr{P}} \mathscr{V}e & \text{if} \quad \mathscr{P} \neq Ⓕ \\ \varnothing & \text{if} \quad \mathscr{P} = Ⓕ \end{cases}$$

and the application of a substitution σ to $\mathscr{P}$ by

$$\sigma\mathscr{P} = \begin{cases} \{\!\{\sigma e, e'\rangle \mid \langle e, e'\rangle \in \mathscr{P}\}\!\} & \text{if} \quad \mathscr{P} \neq Ⓕ \\ Ⓕ & \text{if} \quad \mathscr{P} = Ⓕ. \end{cases}$$

If S is a subset of $\Sigma(\mathscr{F},\mathscr{V})/\mathscr{A}$, σ a substitution and $\mathscr{W}$ a subset of $\mathscr{V}$, we define $S \circ \sigma = \{\rho \circ \sigma \mid \rho \in S\}$ and $S \restriction \mathscr{W} = \{\rho \restriction \mathscr{W} \mid \rho \in S\}$.

https://doi.org/10.1017/S0960129505004883 Published online by Cambridge University Press

Finally, let us recall some definitions and results about monoids that will be needed.

Let $\mathscr{V}$ be a set of variables, $\mathscr{C}$ be a set of constants (that is, nullary function symbols), $\star$ and $\wedge$ two extra function symbols with respective arities 2 and 0. The *free monoid* over $\mathscr{V} \cup \mathscr{C}$ is denoted $(\mathscr{V} \cup \mathscr{C})^\star$ and is the initial algebra $\mathrm{M}\left(\mathscr{C} \cup \{\star, \wedge\}, \mathscr{V}\right)/\mathscr{A}_\star$ where $\mathscr{A}_\star$ is the following set of axioms:

$$\mathscr{A}_\star : \begin{cases} \mathscr{A}_{\star 1} : (x \star y) \star z = x \star (y \star z) & \text{associativity} \\ \mathscr{A}_{\star 2} : x \star \wedge = x & \text{right identity} \\ \mathscr{A}_{\star 3} : \wedge \star x = x & \text{left identity.} \end{cases}$$

The elements of $(\mathscr{V} \cup \mathscr{C})^\star$ are called *words* or *sequences*. The variable set of a word is that of any of its representations. The size of a word u is an integer, denoted $|u|$, and defined by:

$$|u| = \begin{vmatrix} 0 & \text{if} & u = \wedge \\ 1 & \text{if} & u = @ \text{ and } @ \in \mathscr{V} \cup \mathscr{C} \\ |u'| + |u''| & \text{if} & u = u' \star u''. \end{vmatrix}$$

There is a procedure, called here *MATCHWORD*, which takes as argument a single matching pair $\langle u, u' \rangle$ on $(\mathscr{V} \cup \mathscr{C})^\star$ and returns a set of matches for that pair (Plotkin 1972; Siekmann 1975). The procedure *MATCHWORD* always terminates; it is consistent, complete and minimal, and for every match ω produced by *MATCHWORD* $\left(\langle u, u' \rangle\right)$, we have $\mathscr{V}\omega = \varnothing$ and $\mathscr{D}\omega \subseteq \mathscr{V}u$.

3. An equational theory with multiplication and exponentiation

Let $\mathscr{C}$ be a denumerable set of constants and $\mathscr{V}$ a denumerably infinite set of variables. Let $\times$, $\uparrow$ and 1 be three function symbols with respective arities 2, 2 and 0. We consider the set $\mathscr{F} = \mathscr{C} \cup \{\times, \uparrow, 1\}$ of function symbols.

We use $\mathscr{T}$ to denote the free algebra $\mathrm{M}(\mathscr{F}, \mathscr{V})$.

Let $\mathscr{A}$ be the following system of axioms:

$$\mathscr{A} : \begin{cases} \mathscr{A}_1 : (x \times y) \times z = x \times (y \times z) & \text{associativity} \\ \mathscr{A}_2 : (x \times y) \uparrow z = (x \uparrow z) \times (y \uparrow z) & \text{left distributivity} \\ \mathscr{A}_3 : x \uparrow (y \times z) = (x \uparrow y) \uparrow z & \text{right currying} \\ \mathscr{A}_4 : x \times 1 = x \\ \mathscr{A}_5 : 1 \times x = x \\ \mathscr{A}_6 : x \uparrow 1 = x \\ \mathscr{A}_7 : 1 \uparrow x = 1. \end{cases}$$

We are interested in the equational theory $\mathscr{E} = \mathrm{M}(\mathscr{F}, \mathscr{V})/\mathscr{A}$.

We use $\mathscr{S}$ to denote the set $\Sigma(\mathscr{F}, \mathscr{V})/\mathscr{A}$ of substitutions on $\mathscr{E}$.

The general definition of the variable set of an expression $e \in \mathscr{E}$ as $\mathscr{V}e = \bigcap_{t \in e} \mathscr{V}t$ is not computationally usable. Thus we give an equivalent definition of $\mathscr{V}e$ by means of a

https://doi.org/10.1017/S0960129505004883 Published online by Cambridge University Press

morphism ψ from $\mathscr{E}$ to $2^{\mathscr{V}\cup\mathscr{C}}$ defined roughly as follows:

$$\psi e = \begin{cases} \{@\} & \text{if} \quad e = @ \text{ and } @ \in \mathscr{V} \cup \mathscr{C} \\ \varnothing & \text{if} \quad e = 1 \\ \psi e' \cup \psi e'' & \text{if} \quad e = e' \times e'' \\ \varnothing & \text{if} \quad e = e' \uparrow e'' \text{ and } \psi e' = \varnothing \\ \psi e' \cup \psi e'' & \text{if} \quad e = e' \uparrow e'' \text{ and } \psi e' \neq \varnothing. \end{cases}$$

We prove that ψ is a well-defined morphism of $\mathscr{F}/\mathscr{A}$-algebras, and that for every expression e in $\mathscr{E}$ we have $\mathscr{V} e = \psi e \cap \mathscr{V}$.

As a corollary, we get

$$\begin{array}{ll} \forall x \in \mathscr{V}. & \mathscr{V}\,[x]_{\mathscr{A}} = \{x\} \\ \forall a \in \mathscr{C} \cup \{1\}. & \mathscr{V}\,[a]_{\mathscr{A}} = \varnothing \\ \forall e, e' \in \mathscr{E}. & \mathscr{V}(e \times e') = \mathscr{V} e \cup \mathscr{V} e' \text{ and } \mathscr{V} e \subseteq \mathscr{V}(e \uparrow e'). \end{array}$$

The usual concepts of *base* and *exponent* are generalised to all expressions in $\mathscr{E}$.

We define the base mapping B as a morphism of $\mathscr{F}/\mathscr{A}$-algebras from $\mathscr{E}$ to $(\mathscr{V} \cup \mathscr{C})^\star$ as follows:

$$Be = \begin{cases} @ & \text{if} \quad e = @ \text{ and } @ \in \mathscr{V} \cup \mathscr{C} \\ \wedge & \text{if} \quad e = 1 \\ Be' \star Be'' & \text{if} \quad e = e' \times e'' \\ Be' & \text{if} \quad e = e' \uparrow e''. \end{cases}$$

We remark that the monoid $(\mathscr{V} \cup \mathscr{C})^\star$ may be considered an $\mathscr{F}/\mathscr{A}$-algebra by choosing the left projection $\lambda XY.X$ for exponentiation.

The exponent mapping E is defined by means of a morphism ϕ from $\mathscr{E}$ to the cartesian product $\mathscr{E}^\star \times \mathscr{E}$ structured as an $\mathscr{F}/\mathscr{A}$-algebra.

The morphism ϕ is defined as follows:

$$\phi e = \begin{cases} \langle 1, @ \rangle & \text{if} \quad e = @ \text{ and } @ \in \mathscr{V} \cup \mathscr{C} \\ \langle \wedge, 1 \rangle & \text{if} \quad e = 1 \\ \langle \phi_1 e' \star \phi_1 e'', \phi_2 e' \times \phi_2 e'' \rangle & \text{if} \quad e = e' \times e'' \\ \langle \phi_1 e' \vec{\times} \phi_2 e'', \phi_2 e' \uparrow \phi_2 e'' \rangle & \text{if} \quad e = e' \uparrow e'' \end{cases}$$

where $\phi_i e$ is the i-th component of ϕe, and $\vec{\times}$ is the extension to $\mathscr{E}^\star$ component-wise on the left of the multiplication $\times$ in $\mathscr{E}$.

The exponent mapping E is then defined as the first component of $\phi : \forall e \in \mathscr{E}.\ Ee = \phi_1 e$.

The base Be and exponent Ee of an expression e are sequences in $(\mathscr{V} \cup \mathscr{C})^\star$ and $\mathscr{E}^\star$, respectively. We use $B_i e$ and $E_i e$ to denote their i-th components.

We easily show that for any expression e in $\mathscr{E}$, the base and exponents of e have the same length, that is, $|Be| = |Ee|$. We then show that e can be decomposed into a multiplication as follows:

$$\forall e \in \mathscr{E}.\ e = \prod_{i=1}^{|Be|} ([B_i e]_{\mathscr{A}} \uparrow E_i e).$$

Thus any expression e is uniquely characterised by its base and exponent.

https://doi.org/10.1017/S0960129505004883 Published online by Cambridge University Press

The concept of a base is extended to substitutions in $\mathscr{S} = \Sigma(\mathscr{F}, \mathscr{V})/\mathscr{A}$. The base $B\sigma$ of a substitution σ in $\mathscr{S}$ is an endomorphism of the free moinoid $(\mathscr{V} \cup \mathscr{C})^\star$. It is defined by

$$\forall \sigma \in \mathscr{S}.\ \forall x \in \mathscr{V}.\ \ (B\sigma)x = B(\sigma x).$$

We prove that for any substitution σ in $\mathscr{S}$ we have $\mathscr{D}(B\sigma) \subseteq \mathscr{D}\sigma$, and thus $B\sigma$ is a substitution on $(\mathscr{V} \cup \mathscr{C})^\star$.

Other useful properties of base and exponent are

$$\forall \sigma \in \mathscr{S}.\ \forall e \in \mathscr{E}.\ \left| \begin{array}{l} (a)\ \ B \circ \sigma = (B\sigma) \circ B \\ (b)\ \ \mathscr{V}(Be) = \varnothing \implies B(\sigma e) = Be \text{ and } E(\sigma e) = \sigma(Ee) \\ (c)\ \ \bigcup\limits_{x \in \mathscr{V}(Be)} \mathscr{V}(\sigma x) \subseteq \mathscr{V}(\sigma e). \end{array} \right.$$

The notation $\sigma(Ee)$ in (b) implies that the substitution σ on $\mathscr{E}$ has been extended (component-wise) to sequences in $\mathscr{E}^\star$.

The *size* of expressions in $\mathscr{E}$ is defined as a partial function from $\mathscr{E}$ to $\mathbb{N}$ that is the least fix-point of the recursive equation

$$\|e\| \ \Rightarrow\ |Be| + \sum_{i=1}^{|Be|} \|E_i e\|$$

where $\|e\|$ denotes the size of the expression e.

We prove that *size* is in fact a total function and satisfies the following conditions for every expression e:

$$\|e\| = \left| \begin{array}{ll} 0 & \text{if} \quad e = 1 \\ 1 & \text{if} \quad e = @ \text{ and } @ \in \mathscr{V} \cup \mathscr{C} \\ \|e'\| + \|e''\| & \text{if} \quad e = e' \times e'' \\ (|Be'| \times \|e''\|) + \|e'\| & \text{if} \quad e = e' \uparrow e''. \end{array} \right.$$

If $\mathscr{P}$ is a finite set of matching pairs on $\mathscr{E}$, the size of $\mathscr{P}$ is denoted $\|\mathscr{P}\|$ and defined by

$$\|\mathscr{P}\| = \left| \begin{array}{ll} |\mathscr{P}| + \sum\limits_{\langle e, e' \rangle \in \mathscr{P}} \|e'\| & \text{if} \quad \mathscr{P} \neq \text{Ⓕ} \\ 0 & \text{if} \quad \mathscr{P} = \text{Ⓕ} \end{array} \right.$$

where $|\mathscr{P}|$ is the cardinality of the set $|\mathscr{P}|$.

The size of sets of matching pairs is used to prove termination of the matching algorithm.

Finally, given a word substitution ω on $(\mathscr{V} \cup \mathscr{C})^\star$ such that $\mathscr{V}\omega = \varnothing$ and a finite subset $\mathscr{W}$ of $\mathscr{V}$, we define the most general substitution with base ω and variables not in $\mathscr{W}$, denoted $\sigma_{\omega,\mathscr{W}}$, by

$$\sigma_{\omega,\mathscr{W}} = \left\{ \left\langle x_i, \prod_{j=1}^{|u_i|} (u_{i,j} \uparrow x_{i,j}) \right\rangle \ \middle|\ \begin{array}{l} x_i \in \mathscr{D}\omega \text{ and } u_i = \omega x_i \text{ and} \\ \text{the } x_{i,j}\text{'s are distinct variables in } \mathscr{V} - \mathscr{W} \end{array} \right\}$$

where $u_{i,j}$ denotes the j-th component of the word u_i of $\mathscr{C}^\star$, while $x_{i,j}$ is just a variable in $\mathscr{V} - \mathscr{W}$.

https://doi.org/10.1017/S0960129505004883 Published online by Cambridge University Press

This does not define $\sigma_{\omega,\mathscr{W}}$ uniquely, but we only need to assume that we have some unique way of choosing the variables $x_{i,j}$ so that $\sigma_{\omega,\mathscr{W}}$ is uniquely determined.

Naturally, we have $B\sigma_{\omega,\mathscr{W}} = \omega$, $\mathscr{D}\sigma_{\omega,\mathscr{W}} = \mathscr{D}\omega$ and $\mathscr{V}\sigma_{\omega,\mathscr{W}} \cap \mathscr{W} = \varnothing$, but the main property of $\sigma_{\omega,\mathscr{W}}$ is that for any substitution σ in S we have

$$\omega \leqslant B\sigma \implies \exists \sigma' \in S.\ \sigma \restriction \mathscr{W} = (\sigma' \circ \sigma_{\omega,\mathscr{W}}) \restriction \mathscr{W}.$$

4. The matching algorithm

The matching algorithm is composed of a main procedure *MATCH* and a subprocedure *SIMPL*. The procedure *MATCH* uses the word matching procedure *MATCHWORD* to match the bases of matching pairs; it calls itself recursively in order to also match the exponents. The procedure *SIMPL* just reduces the matching of a pair $\langle e, e' \rangle$ to the matching of its exponents whenever the bases are equal and without variables.

procedure *SIMPL* ($\mathscr{P}$);
% *The parameter* $\mathscr{P}$ *is a set of matching pairs on* $\mathscr{E}$. %
% *The result of SIMPL is a new set, say* $\mathscr{P}''$, *of matching pairs such that:* %
% $\mathscr{M}(\mathscr{P}) = \mathscr{M}(\mathscr{P}'')$ *and* $\forall \langle e, e' \rangle \in \mathscr{P}''.\ \mathscr{V}(Be) \neq \varnothing$ %
begin
Choose a pair $\langle e, e' \rangle$ in $\mathscr{P}$ such that $\mathscr{V}(Be) = \varnothing$;
if there is no such pair **then** *SIMPL* := $\mathscr{P}$
% *this is in particular the case when* $\mathscr{P} = \textcircled{F}$ %
else **if** $Be \neq Be' =$ **then** *SIMPL* := $\textcircled{F}$
else **begin**
$\mathscr{P}'$:= $(\mathscr{P} - \langle e, e' \rangle) \cup \{\!\{ E_i e, E_i e' \rangle \mid 1 \leqslant i \leqslant |Be| \}\!\}$;
SIMPL := *SIMPL* $(\mathscr{P}')$;
end ;
end ;
procedure *MATCH* ($\mathscr{P}$) ;
% *The parameter* $\mathscr{P}$ *is a set of matching pairs on* $\mathscr{E}$. %
% *The result of MATCH is a set of matches of* $\mathscr{P}$. %
begin
$\mathscr{P}'$:= *SIMPL* ($\mathscr{P}$) ;
$\mathscr{W}$:= $\mathscr{V}\mathscr{P}'$;
if $\mathscr{P}' = \varnothing$ **then** *MATCH* := $\{\varnothing\}$ % *return identity match only* %
else **if** $\mathscr{P}' = \textcircled{F}$ **then** *MATCH* := $\varnothing$ % *return no match* %
else **begin**
Choose a pair $\langle e, e' \rangle$ in $\mathscr{P}'$;
Ω := *MATCHWORD* $(\langle Be, Be' \rangle)$;
MATCH := $\bigcup_{\omega \in \Omega} (MATCH\ (\sigma_{\omega,\mathscr{W}}\mathscr{P}') \circ \sigma_{\omega,\mathscr{W}}) \restriction \mathscr{W}$;
end ;
end ;

https://doi.org/10.1017/S0960129505004883 Published online by Cambridge University Press

We prove the following results:

— The procedures *SIMPL* and *MATCH* always terminate with finite input $\mathscr{P}$.
— The procedure *SIMPL* is correct: $\mathscr{M}\mathscr{P} = \mathscr{M}(SIMPL\ (\mathscr{P}))$.
— The procedure *MATCH* is *consistent*, *complete* and *minimal*.

We also establish that, for any substitution σ in $MATCH\ (\mathscr{P})$ we have $\mathscr{V}\sigma = \varnothing$ and also $\forall x \in \mathscr{D}\sigma.\ \sigma \restriction (\mathscr{V} - \{x\}) \notin \mathscr{M}\mathscr{P}$.

In the final section we use the minimality of *MATCHWORD* (which is not necessary for the above results) to prove that no substitution is computed twice by the procedure *MATCH*. We then consider variants of the algorithm with respect to efficiency of implementation.

Acknowledgements

I wish to thank Mikael Rittri for making this work better known, and Roberto Di Cosmo for this opportunity to have it published.

References

Darlington, J. and Burstall, R. (1973) A system which automatically improves programs. *Proc. 3rd IJCAI*, Stanford, California, USA, 479–485. (Also: *Acta Informatica* (1976) **6** 41–60.)

Huet, G. (1975) A unification algorithm for typed λ-calculus. *Theoretical Computer Science* **1** (1) 27–58.

Kühner, S., Mathis, C., Raulefs, P. and Siekmann, J. H. (1977) Unification of idempotent functions. *Proc. 5th IJCAI*, Cambridge, Massachusetts, USA **1** 528.

Paterson, M. S. and Wegman, M. N. (1976) Linear unification. *Proc. 8th ACM Symp. on Theory of Computing* 181–186.

Plotkin, G. D. (1972) Building-in equational theories. In: Meltzer, B. and Michie, D. (eds.) *Machine Intelligence* **7** 73–90.

Robinson, J. A. (1965) A machine-oriented logic based on the resolution principle. *J. ACM* **12** (1) 23–41.

Siekmann, J. H. (1975) String unification. Essex University Memo CSM-7.

Slagle, J. R. (1974) Automated theorem-proving for theories with simplifiers, commutativity, and associativity. *Journal of the ACM* **21** (4) 622–642.

Stickel, M. (1975) A complete unification algorithm for associative-commutative functions. *Proc. 4th IJCAI*, Tbilisi, Georgia, USSR **1** 71–76.

Scott, D. (1977) Lecture Notes from the École Avancée de Sémantique, Sophia-Antipolis, France.

Historical Note

This paper reproduces an extended abstract that was submitted to conferences in 1978, without success. At the time, the author was busy with other projects and could not afford the time to get a full version typed (on a typewriter), given the apparent lack of interest. Complete proofs exist in handwritten notes that were never circulated, and this extended abstract is the only document that has been made available to interested people. The rule

followed for publication here was that the document should remain exactly as originally circulated. The author is aware that, given this constraint and the original intent of the document, the style does not meet the expected standards of journal publication, and he wishes to apologise for this to the reader. The only changes have been the typographic adaptation to MSCS style, the correction of minor typos and slightly more precision in the references, with no change to the list of referenced documents.

Though criticised for it, this paper made the deliberate choice of not using normal forms. It was felt that the use of normal forms presupposes that term structures (that is, trees) are the most appropriate representations for equivalence classes of terms. However, there is no reason to believe that this is always true. Defining mappings by means of morphisms did suggest carrying computations on term structures, but was actually intended as a simple mean of ensuring that they are well defined and effectively computable by structural recursion.

The motivation of the author was to extend the work in Huet and Lang (1978)[†]. This work proposes the use of second-order matching of λ-terms to identify applicable program transformation schemes. The author realised that being able to apply the transformations up to type isomorphism would significantly extend their useful range while allowing a reduction in the size of the transformation library. Matching their types was perceived by the author as a first step towards matching the λ-expressions. The intent was later to extend the λ-expression matching algorithm by mixing it with type matching (followed by the corresponding conversions of the pattern) to guide λ-expression matching modulo type isomorphism.

[†] Huet, G. and Lang, B. (1978) Proving and applying program transformations expressed with second order patterns. *Acta Informatica* **11** 31–55.